소프트웨어 모델링 및 분석

OOAD vs SASD

Team 4

200611450 강세용

200611458 김영승

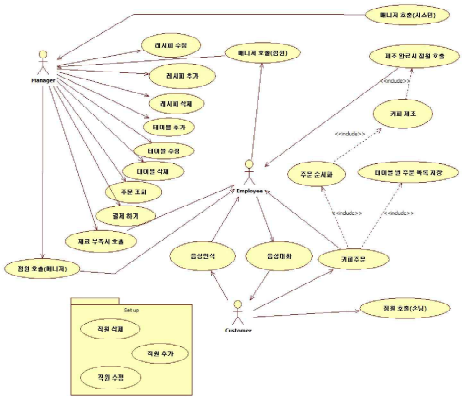
200611518 조민경

1. OOAD

1.1. 1st OSP1000

- 말이 없는 커피메이커 시스템을 만들기 위해, 처음 시스템을 크게 세가지, Controller, Server, CoffeeMaker로 나누어 OOAD를 진행하였다.

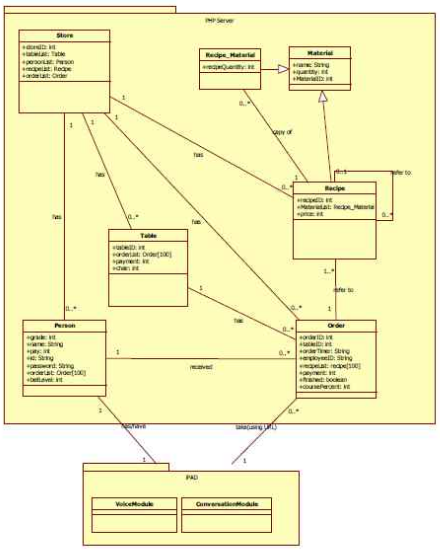
그래서 나온 OOAD를 진행하여 나온 첫번째 UseCase 이다.



하지만 위 그림을 보시다시피 OOAD 1000 단계에서 나온 usecase의 경우 시스템의 경계가 명확하게 나뉘지 않아 문제가 되다. 어떤 usecase가 어떤 시스템에 속해있는지, 또는 어떠한 actor가 동작을 수행하는지에 대한 명확한 구분이 되지 않아 잘 작성된 usecase라고는 할 수 없다.

1.2. 2nd OSP2130

- 그 후 두 번째 단계에서 다시 usecase diagram을 재정의 하였으며 이로써 컨트롤러와 서버, 커피메이커의 경계가 명확해지게 되었다. 이 단계에서의 usecase는 Database와 coffee maker를 extra actor로 분류하여 server와 controller에 더욱 초점을 맞추었다.

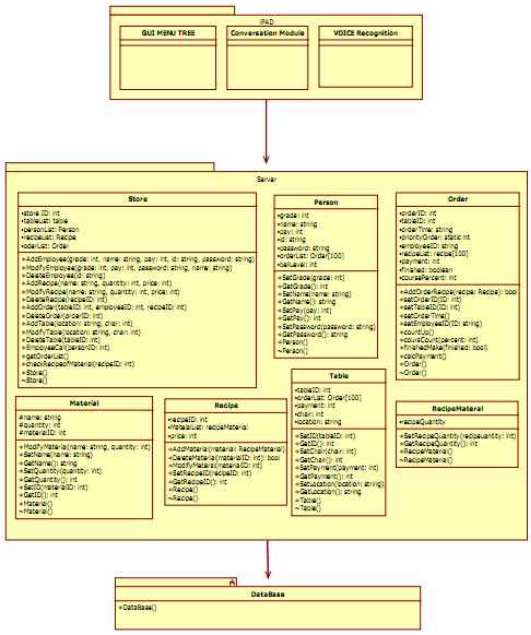


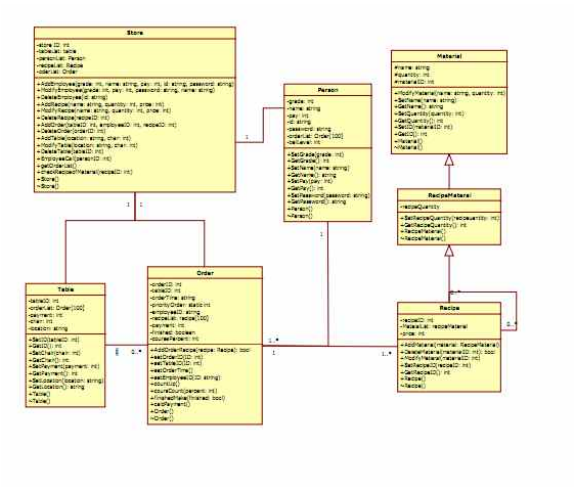
그렇게 작성하여서 나온 Deployment Diagram 이다. 이로써 각 Server와 iPad에 들어갈 간단한 class object들을 정의하고 대략적으로 어떤 시스템이 될지를 그려볼 수 있었다. 또

한 각 Usecase 마다 시퀀스 다이어그램을 그려봄으로써 서로의 시스템이 어떻게 서로에게 영향을 주고 어떻게 동작하는지에 대한 개괄적인 내용을 그려볼 수 있었다.

1.3. 3rd OSP2140

- 위에서 그린 Deployment Diagram과 system들의 시퀀스 다이어그램을 바탕으로 정의된 System Architecture이다.





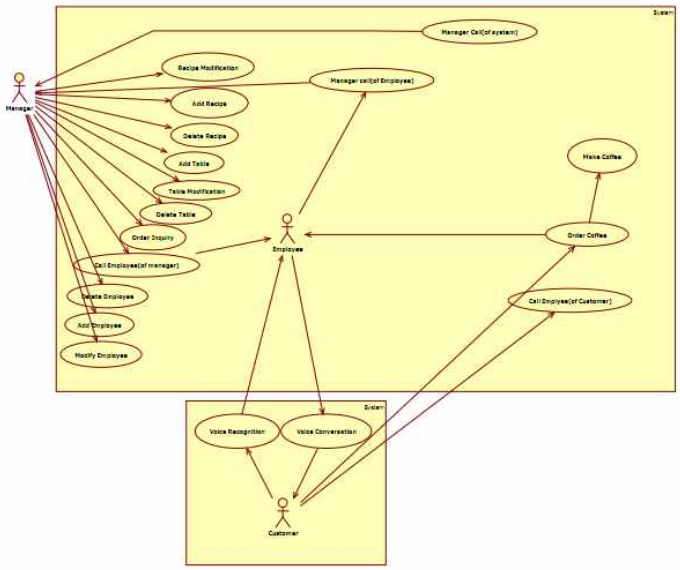
<<System Architecture>>　　<<Interaction Diagram>>

그 다음 클래스간의 상호작용과 연관관계를 알아보기 위해 interaction Diagram들을 그렸으며 이로써 각 클래스간의 동작과정과 내부 함수, 변수들을 좀 더 명확히 정의하고 만들어 볼 수 있었다.

이런 모든 과정을 끝내고 나온 클래스 다이어그램을 최종적으로 Code Generation을 하였으며 이로써 OOAD의 첫 번째 싸이클을 끝낼 수 있었다.

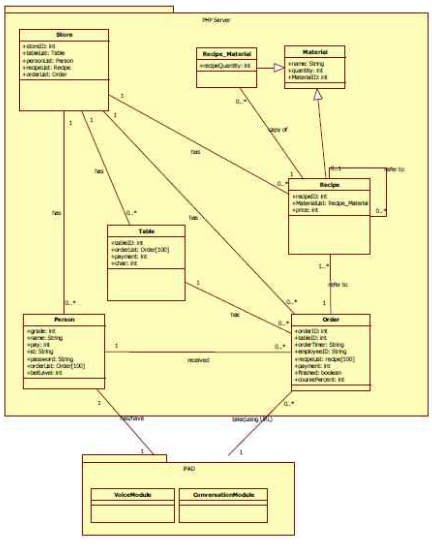
1.4. 4th OSP2200

- 두 번째 cycle에서 중요하게 생각한 것은 첫 번째 진행할 때 명확하게 정의하지 못한 usecase를 좀 더 명확하게 하는 것이었다.

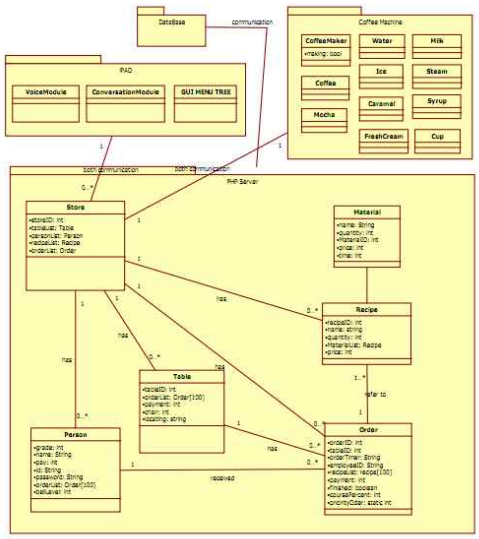


그리고 첫 번째 cycle에서 extra actor로 구분했던 coffee maker를 좀 더 자세하게 정의하였다.

<1st cycle>



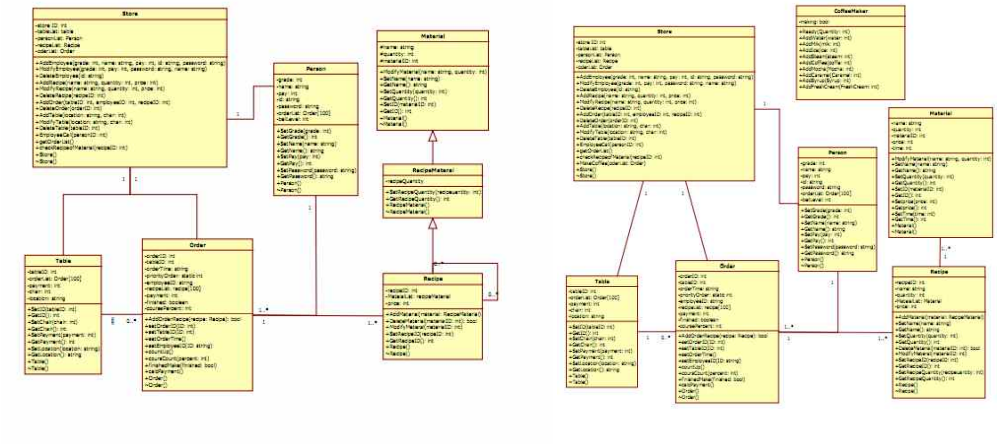
<2ed cycle>



그렇게 나온 Domain Model이다. 첫 번째 cycle과는 다르게 extra actor였던 DB의 위치를 좀더 명확하게 하였으며 CoffeeMaker의 내부 모듈을 정의하여 server와 controller, coffee maker의 연결관계를 더 명확하고 정확하게 정의하였다.

이렇게 재정의된 다이어그램들을 토대로 다시 작성된 두 번째 cycle의 최종 다이어그램이다.

<1st cycle> <2ed cycle>

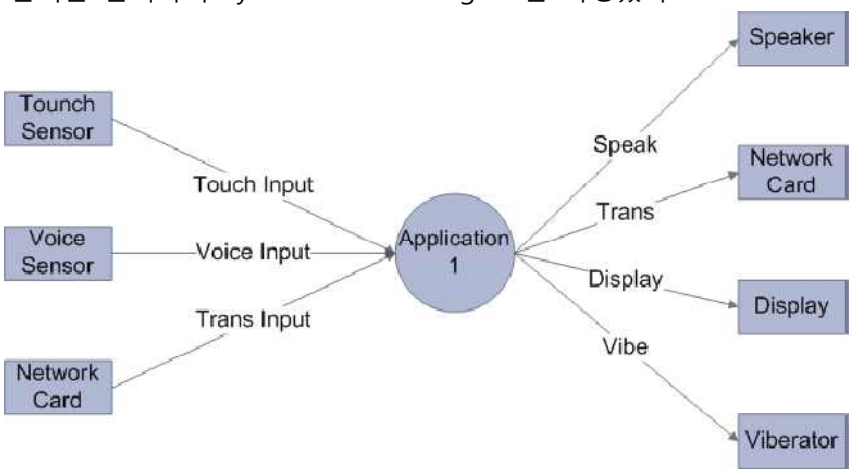


아주 커다란 변화는 없었지만, 세부적인 변수들과 함수들 그리고 복잡하게 나뉘어져 있었던 클래스다이어그램들이 간단하게 변화하였다. 이렇게 두 번째 cycle을 끝내고 OOAD의 진행을 마무리하였다.

2. SASD

2.1. SA

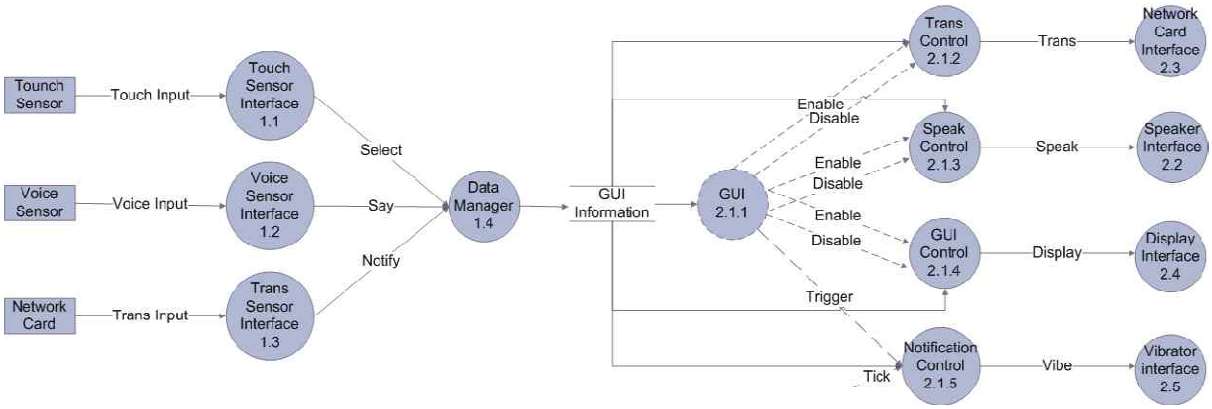
- SA에서는 Top-Down 방식의 Divide and Conquer 방법을 사용한다. 말이 필요 없는 커피메이커 시스템은 컨트롤러인 iPad와 실제 프로세스가 실행되는 Server, 그리고 커피메이커의 서브시스템으로 나누어진다. 각 서브시스템 별로 SASD를 진행하였다. 먼저 시스템의 외부 입력과, 출력을 분석하여 System Context Diagram을 작성했다.



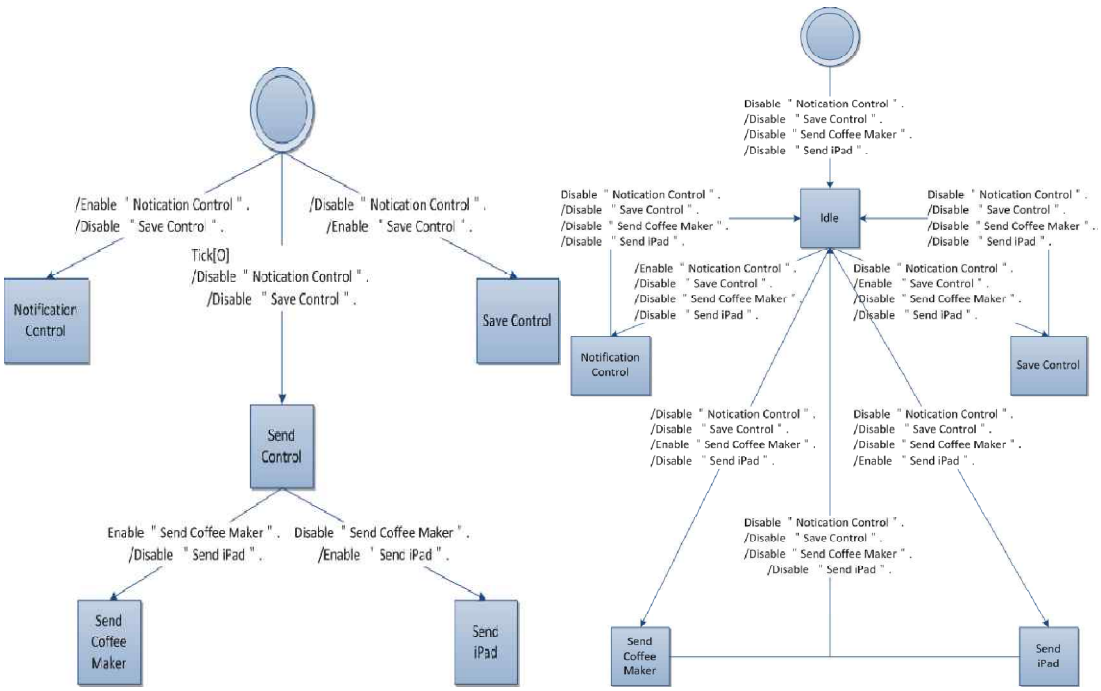
System Context Diagram에서 나온 시스템 외부의 입력과 출력의 Event List를 작성한다. iPad에서는 Touch input, Voice input, Trans input의 입력과 Speak, Trans, Display, Vibrate의 출력으로 나누어지고 프로세스가 입력을 처리하여 결과를 출력하게 된다.

DFD level 0에서는 전체 시스템의 외부 입력과 출력, 입력을 처리하는 프로세스가 표현된다. iPad에서는 외부 센서에서의 입력을 받아 Data Manager프로세스가 GUI Information 정보로 가공하고 GUI 프로세스가 이 정보를 받아 외부로 출력한다.

DFD level 0이 작성되면, 프로세스를 분석하여 프로세스 내부의 데이터 흐름과 내부 데이터 처리를 위한 프로세스들로 나눈다. 프로세스가 더 이상 나누어지지 않을 때 가지 나눠 DFD를 단계별로 작성한다. 최종적으로 나눠진 DFD를 종합하여 전체 시스템의 DFD를 구할 수 있다.

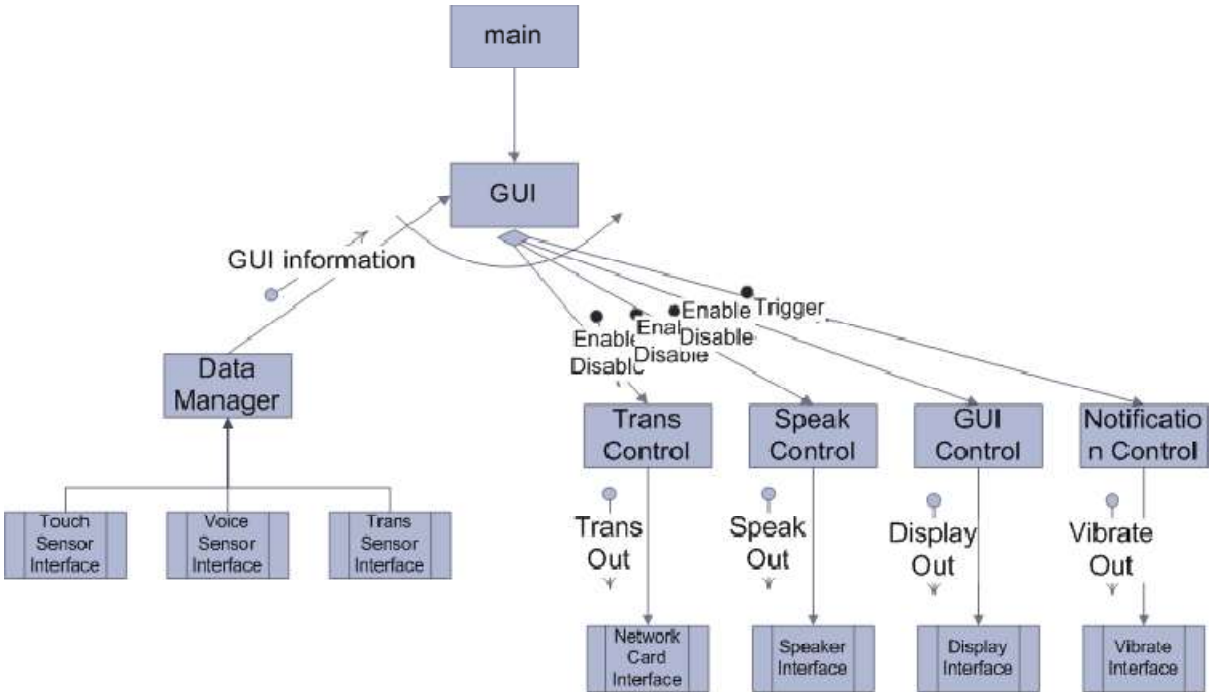


전체 DFD를 참조하여 시스템의 상태와 상태 변화를 나타내는 FSM를 작성한다. FSM을 작성하는 과정에서 각 상태가 종료된 후 다시 돌아가는 상태가 없어 수정을 하였다. 기본적으로 FSM은 시스템의 각 상태와 상태의 변화를 나타내는 것이기 때문에 시스템이 종료되는 경우가 아니라면 상태가 종료된 후 다시 초기상태로 돌아가도록 작성도어야 한다.



2.2. SD

- SA에서 나온 DFD를 기준으로 Structred Charts를 작성한다.



GUI 프로세스를 기준으로 왼쪽의 입력과, 오른쪽의 출력으로 나눠 표현했다. 전체적인 DFD Diagram을 분석한 결과 모든 모듈은 자신의 동작을 수행한 뒤에 idle상태로 돌아가는 형식이기 때문에 Main에서 for문을 통해 반복하는 형태로 구성했다. Structured Chart를 작성하게 되면 해당 시스템에 대한 구체적인 코드형태가 나타나게 돼서 좀 더 가시적인 시스템을 표현할 수 있었다. 이것은 다음 단계인 Pseudo Code를 작성하는데 큰 도움이 되었다.

Structured chart를 작성한 후 Pseudo Code를 작성하였는데 Structured Chart만을 가지고 Pseudo Code를 작성하기에는 힘들었다. 그래서 SA단계에서 작성하였던 FSM을 같이 참고하여 작성하였다. Pseudo Code를 작성하는 것은 생각보다 적은 시간이 걸렸다. 이미 Structured chart에서 Code에 대한 전반적인 구성이 보이기 때문에 큰 어려움 없이 진행할 수 있었다.

3. OOAD vs SASD

3.1. Adv. Disadv.

- OOAD는 객체지향 분석 설계 방법론으로써, Use Case를 통해 전체적인 분석과 설계가 진행된다고 할 수 있다. 먼저, 개발하고자 하는 시스템의 요구사항을 통하여 시스템에 대한 정의를 하게 된다. 이 정의를 통해 추상적인 Use Case를 뽑아내고, 이 Use Case를 각 단계마다 구체화시키고 재정의하면서 전체적인 시스템의 모습을 만들어 낸다고 할 수 있다. 따라서, 초기 단계에서 잘못된 분석은 다음 단계에 영향을 미치게 되고 이로 인해 전체적인 수정을 하게 되기도 한다. 실제 OOAD 개발 방법론을 사용한 이번 프로젝트에서 초기에 잘

못된 시스템 분석과 Use Case 분석으로 인해서 다시 한 번 분석을 하기도 하였다.

이렇게 분석된 Use Case를 통해서 전체적인 시스템의 구성을 하게 되고, 각 구성원은 각각 하나의 Class가 되어 다른 Class와 상호작용하게 된다. 이 상호작용을 통하여 전체적인 시스템의 모습이 가시화 되고, 마지막으로 전체적인 클래스간에 상호작용에 대한 흐름을 작성하게 된다.

앞에 말했듯이, OOAD는 앞 단계의 결과물이 후반부와 전혀 상관없이 진행되는 것이 아니라 단계적으로 확장되는 방식으로 기본적인 흐름에는 Use Case가 존재하게 된다. 따라서, 초기 분석이 잘 되었다면 자연스럽게 후반부의 설계까지 직관적으로 진행이 가능하지만 반대의 경우도 존재하게 된다. 전체적인 흐름으로 보면 하나의 건축과 비교할 수 있다. 초기 공사가 잘 되어야 전체적인 건축물이 안정적일 수 있는 것과 마찬가지 인 것이다.

이에 비해 SASD 방법론은 Divide and Conquer와 Top-Down으로 표현할 수 있는 방법론이다. 전체적인 시스템을 하부 시스템으로, 이 하부시스템을 하나하나 분석하며 아래 단계로 차근차근 내려가게 되는 것이다. 소프트웨어적인 접근이라고 하기보다는 하드웨어 대한 접근에 조금 더 가깝다고 생각할 수 있다. 전체적인 시스템의 input과 output의 정의로 시작하는 SASD에서는 각 시스템 별로 별도의 분석을 실시한다. 이 분석으로 각 모듈을 정의하게 되고 모듈의 내용을 작성하게 된다. 이 모듈은 프로세스와 컨트롤러로 표현되게 된다.

SASD에서는 전체적인 모듈을 나누어 정의하는 과정에서 직관적이라는 장점을 가질 수 있다. 하나가 들어가서 하나가 나온다는 One-in One-out 구성을 기본으로 하기 때문에, 사람이 생각하는 것과 유사한 방법이라고 할 수 있는 것이다. 하지만, 이 직관성은 동시에 잘못된 분할을 야기할 수도 있다. 이 잘못된 분할은 너무 작은 단위로 분할을 하여 쓸데 없는 프로세스가 생성이 되거나, 너무 큰 단위로 남겨두어 프로세스나 컨트롤러의 역할을 하지 못하는 상황이 발생하는 것이다. 마치 음식을 만들 때 재료의 크기가 너무 작으면 자체 맛을 제대로 내지 못하고, 너무 크게 하면 먹기에 불편한 것과 유사하다고 할 수 있다.

이 접근 방식은 OOAD와는 차이가 크기 때문에 OOAD라는 방법론을 통하여 한 번 작성한 시스템을 다시 작성하는 과정에서 전체적인 개념의 재정립이 힘들었다. OOAD에서 클래스의 개념이 남아있기 때문에 SASD에 식별되어야 하는 프로세스나 컨트롤러와 연관되어 생각을 했기 때문이다. 초기에 각각의 프로세스에 대한 분할에서 클래스 단위의 그림이 아직 머리 속에 남아 있어서, 프로세스를 하나의 클래스 단위로 처리하려고 하는 오류를 범하기도 했다.

3.2. 효과적인 방법론 적용

- 각각의 방법론에는 장단점이 존재하기 때문에 어떤 시스템의 작성에 어떤 방법론을 적용하느냐에 따라 차이가 발생한다. OOAD의 경우는 각각의 객체간의 상호작용에 의해 시스템을 정의하기 때문에 대형 소프트웨어나 다양한 서비스를 제공하는 시스템에 적합하다고 할 수 있다. 대형 소프트웨어의 경우 시스템이 해야 하는 일이 다양하기 때문에 그 다양성을 위주로 분석하는 것이 시스템 전체를 Top-down 해 가는 것보다는 더 효율적일 수 있다.

  반대로 SASD 같은 경우는 소형 소프트웨어, 특히 하드웨어 위주의 작성에 더 용이 하다고 할 수 있다. 물론, 단순 소프트웨어를 작성할 수 없는 것은 아니지만 하드웨어 중심 시스템에서 좀 더 탁월하게 사용할 수 있다는 것이다.

4. 결론

- 이번 소프트웨어 모델링 및 분석 수업을 통해 실제 시스템을 OOAD와 SASD 방법론에 따라 작성을 하였다. OOAD의 경우 작년 2학기에 진행하였던, 소프트웨어 공학 수업에서 한 번 진행하였기 때문에 전체적으로 어려움 없이 프로젝트를 진행 할 수 있었다. 다만 IEEE Standard에 맞춰 진행했던 것과는 OSP라는 실무형 모델을 기반으로 작성하였다는 것에 차이가 있었다. OSP의 경우 실제 모델에 가깝기 때문에 단계의 진행이 좀더 액티브하고 단계별로 진행함에 따라 자연스럽게 최종 설계가 완료되도록 되어 있었다. 또한, UML Tool이라는 정규 표현을 사용하여 작성을 하기 때문에 프로젝트 분업에 있어서도 훨씬 더 쉽게 진행 할 수 있었다.

SASD의 경우에는 초기 작성에 어려움이 많았던 만큼 시간 투자도 많이 하였다. 프로젝트를 완료한 후 생각된 것은 먼저 SASD를 통해 접근했었더라면 좀 더 쉽게 프로젝트를 진행 할 수 있었을 것 같다는 것이었다. 작성이 완료된 SA의 DFD를 확인하고 이해하는데 별다른 어려움이 없었기 때문이다. 다만 SASD 경우, 각 시스템 별로 분업하여 작성을 하더라도 전체 시스템이 하나의 흐름으로 진행되기 때문에 다른 시스템 간의 동기화 문제가 발생하여 모든 작업을 한 번에 하나씩 함께 진행해야 하는 비효율적인 모습이 보였다. 그렇지 않으면 초기의 시스템 정의에서 input과 output을 정의하면서 서로 주고 받아야 하는 데이터에 대한 정의를 먼저 해야 했는데, 시스템 내부 처리 문제가 있었기 때문에 그렇게 하지 못했다.

이번 학기의 프로젝트를 마치면서 OOAD와 SASD 두 개의 방법론에 의한 접근을 하며 모델링에 대한 것을 많이 배우게 되었다. 그리고 느낀 것은 방법론 사이에는 어느 것이 더 우월하다는 것이 없다는 것을 느꼈다. 왜냐하면 OOAD와 SASD 모두 장단점이 존재하기 때문이다. 따라서, 방법론 자체의 문제보다는 어떤 상황에서 어떤 시스템을 작성하느냐에 맞추어 좀 더 효율적인 방법을 선택하는 것이 더 중요한 것이다. 즉, 방법론 자체의 문제가 아니라 어떤 방법론을 사용할 것인가를 선택하는 것이 소프트웨어 모델링 전에 선행되어야 하는 가장 큰 문제가 아닐까 생각한다.